Administracion de Base de Datos

TESH
HUIXQUILUCAN

3.3 Asignación de cuotas de usuario

Creación de base de datos

```
CREATE DATABASE EjemploDB;
```

Creación del Usuario

A continuación, creamos un usuario nuevo en el sistema de gestión de base de datos. Sustituye 'usuario' y 'contraseña' por el nombre de usuario y la contraseña que desees.

```
CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'contraseña';
```

Asignar Privilegios al Usuario

Luego, asignamos privilegios al usuario para que pueda interactuar con la base de datos EjemploDB.

```
GRANT ALL PRIVILEGES ON EjemploDB.* TO 'usuario'@'localhost';
```

Después de asignar los privilegios, es importante ejecutar el siguiente comando para que los cambios tengan efecto:

```
FLUSH PRIVILEGES;
```

Limitar la Cuota de Espacio

MySQL no proporciona un sistema de gestión directo para limitar el espacio en disco utilizado por un usuario de manera nativa. Para gestionar las cuotas de espacio, normalmente tendrías que usar herramientas externas o scripts personalizados que monitoricen y limiten el espacio usado por las tablas asociadas a un usuario.

Una estrategia básica sería:

- **Monitorizar el uso del espacio**: Crear un script o procedimiento almacenado que periódicamente revise el espacio utilizado por las tablas del usuario.
- **Implementar una política de cuotas**: El script puede enviar alertas cuando el espacio de un usuario se aproxime al límite de su cuota y puede restringir la inserción de nuevos datos una vez que se alcance la cuota.

Ejemplo simple de cómo podrías escribir un procedimiento almacenado que verifica el espacio utilizado por las tablas de un usuario específico y bloquea la inserción de datos si se supera un límite específico. Este es un enfoque muy básico y debería ser adaptado y mejorado para producción.

```
DELIMITER //

CREATE PROCEDURE ChequearCuota()
BEGIN
    DECLARE espacio_usado INT;
    SELECT SUM(data_length + index_length) INTO espacio_usado
    FROM information_schema.TABLES
    WHERE table_schema = 'EjemploDB' AND table_owner = 'usuario';

    IF espacio_usado > 1024000000 THEN -- suponiendo una cuota de 1GB
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Límite de cuota excedido';
    END IF;
```

	END // DELIMITER ;

Esta línea cambia el delimitador utilizado por MySQL de ; a //. Esto es necesario para definir procedimientos almacenados que contienen múltiples declaraciones terminadas en ; sin que MySQL interprete el final del procedimiento antes de tiempo.

	DELIMITER //

Esta línea comienza la definición de un nuevo procedimiento almacenado llamado ChequearCuota. Los procedimientos almacenados son rutinas que puedes llamar en tu base de datos para ejecutar operaciones.

	CREATE PROCEDURE ChequearCuota()

Indica el inicio del bloque de código que constituye el cuerpo del procedimiento almacenado.

	BEGIN

Declara una variable local dentro del procedimiento llamada espacio_usado de tipo INT (entero), que se usará para almacenar la cantidad de espacio usado.

	DECLARE espacio_usado INT;

Esta declaración SELECT calcula la suma de data_length y index_length para todas las tablas que pertenecen al esquema 'EjemploDB' y cuyo propietario es 'usuario'. El data_length representa el tamaño total de los datos de la tabla, y el index_length es el tamaño total de los

índices. El resultado se almacena en la variable espacio_usado. information_schema.TABLES es una tabla del sistema que contiene información sobre todas las tablas en la base de datos.

```
SELECT SUM(data_length + index_length) INTO espacio_usado
FROM information_schema.TABLES
WHERE table_schema = 'EjemploDB' AND table_owner = 'usuario';
```

Esta línea es una declaración IF que comprueba si el valor almacenado en espacio_usado excede 1 GB (expresado en bytes).

```
IF espacio_usado > 1024000000 THEN -- suponiendo una cuota de 1GB
```

Si la condición de la declaración IF es verdadera, entonces esta línea envía una señal que es una excepción generada manualmente con un mensaje de error Límite de cuota excedido. El SQLSTATE '45000' representa un estado de error genérico que indica una excepción no manejada.

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Límite de cuota excedido';
```

Marca el fin de la declaración IF.

```
END IF;
```

indica el final del bloque del procedimiento almacenado.

```
END //
```

Restablece el delimitador a su valor por defecto ;, que es el utilizado normalmente en SQL para terminar las declaraciones.

```
DELIMITER ;
```

3.4 Espacios para objetos

Los DBMS se basan en archivos para almacenar datos, y estos archivos, o conjuntos de datos, residen en medios de almacenamiento, o dispositivos. Una buena parte del trabajo del DBA implicará la planificación para el almacenamiento real de la base de datos. Algunas tecnologías de almacenamiento son más adecuadas que otras. Sin embargo, la naturaleza mecánica de la unidad de disco los hace más vulnerables al fracaso de los componentes de otro equipo. Además, las formas en que las unidades de disco son utilizados por las bases de datos pueden hacer que la gestión del almacenamiento impredecibles, como la barra lateral "Modern DBMS de uso de disco“ Puede usarse RAID para mejorar la seguridad de los datos. Para aplicaciones de misión crítica la integridad de los datos puede ser más importante que la disponibilidad de datos. Si el soporte es poco fiable y un fallo de las causas de corrupción de datos, los datos perdidos puede ser más de un problema que el tiempo de inactividad. Es imperativo, por tanto, que las soluciones de almacenamiento de base de datos para protegerlos a toda costa. La recuperación de datos desde medios de almacenamiento lleva mucho más tiempo en completarse que la recuperación de datos desde la memoria caché o la memoria. El rendimiento de la base de datos depende de la entrada y salida a disco. La cantidad de datos almacenados es mayor que nunca antes, y los datos se almacenados por más tiempo. Algunos DBMS permiten al tamaño de los archivos temporales de expandirse y contraerse de forma automática. Dependiendo del tipo y la naturaleza de las operaciones de base de datos en proceso, esta fluctuación puede provocar picos de uso del disco El crecimiento de la capacidad de almacenamiento aumenta aún más la complejidad de la gestión de datos y bases de datos. Muchas organizaciones están implementando nuevas tecnologías de almacenamiento, tales como almacenamiento en red (NAS) y redes de área de almacenamiento (SAN), para ayudar

a controlar la cantidad cada vez mayor de almacenamiento necesario para los usos modernos. La gestión del almacenamiento en el entorno dinámico de hoy es una tarea difícil DBA. Hay muchos problemas de almacenamiento que deben ser resueltos antes de que un DBA pueda crear una base de datos. Uno de los temas más importantes es la cantidad de espacio para permitir la base de datos. El cálculo espacial debe tener en cuenta no sólo tablas, índices, sino también, y dependiendo del DBMS, el registro de transacciones. Cada una de estas entidades probablemente requerirá un archivo separado o conjunto de datos, para el almacenamiento persistente.

En una base de datos, un espacio destinado para objetos puede referirse a varias cosas dependiendo del sistema de gestión de bases de datos (SGBD) específico y la estructura que este maneje. Un ejemplo muy común es en el ámbito de las bases de datos Oracle, donde se utilizan los "tablespaces" para organizar y almacenar los datos. Un tablespace es un contenedor lógico que agrupa estructuras de datos físicas como tablas e índices. Otros SGBD pueden tener conceptos similares, aunque con diferentes nombres o implementaciones.

Vamos a ver cómo se podría implementar y utilizar un espacio para objetos en una base de datos Oracle (un "tablespace"), cómo crear objetos dentro de este y un ejemplo práctico de cómo se usarían en la vida real.

Tipos de Espacio para Objetos

1. Tablas: Las tablas son el tipo de objeto más común en una base de datos. El espacio para almacenar tablas puede ser asignado de manera contigua o distribuida en varios archivos de datos.

2. Índices: Los índices se utilizan para mejorar el rendimiento de las consultas al permitir búsquedas rápidas en los datos. El espacio para almacenar índices puede ser separado del de las tablas o estar integrado en el mismo espacio.

3. Vistas: Las vistas son consultas predefinidas que se almacenan en la base de datos. El espacio para almacenar vistas puede ser mínimo, ya que no almacenan datos en sí mismas, sino que son definiciones de consultas.

4. Procedimientos Almacenados: Los procedimientos almacenados son conjuntos de instrucciones SQL que se guardan en la base de datos para su reutilización. El espacio para almacenar procedimientos almacenados puede ser mínimo en comparación con el de las tablas.

5. Funciones, Triggers y Otros Objetos: Dependiendo del DBMS, puede haber otros tipos de objetos que ocupen espacio en la base de datos, como funciones, triggers, secuencias, etc.

Ejemplo Práctico: Creación de Tablespace y Tablas en Oracle

Creación de un Tablespace: Un tablespace en Oracle se crea con el objetivo de gestionar el almacenamiento físico de los datos bajo una agrupación lógica. Aquí está el SQL para crear un tablespace llamado mi_tablespace:

```
 CREATE TABLESPACE mi_tablespace
DATAFILE 'mi_tablespace.dbf'
SIZE 50M
AUTOEXTEND ON
NEXT 10M
MAXSIZE UNLIMITED;
```

En este comando:

· DATAFILE 'mi_tablespace.dbf': Especifica el nombre del archivo de datos.

· SIZE 50M: Define el tamaño inicial del tablespace.

· AUTOEXTEND ON: Habilita la extensión automática del tablespace.

· NEXT 10M: Define en cuánto debe extenderse el tablespace cuando se llene el espacio disponible.

· MAXSIZE UNLIMITED: Permite que el tablespace crezca sin límite.

Creación de una Tabla en el Tablespace Creado: Una vez que tienes el tablespace, puedes comenzar a crear objetos de base de datos dentro de él. Por ejemplo, para crear una tabla:

```
 CREATE TABLE empleados (
  id NUMBER PRIMARY KEY,
  nombre VARCHAR2(100),
  departamento VARCHAR2(50)

```

```
) TABLESPACE mi_tablespace;
```

Aquí, la tabla empleados se almacenará físicamente en el mi_tablespace.

Uso de la Tabla: Ahora que la tabla está creada, puedes realizar operaciones estándar de SQL sobre ella, como inserciones, consultas y actualizaciones:

```
 INSERT INTO empleados (id, nombre, departamento) VALUES (1, 'Juan Pérez', 'Recursos Humanos');
SELECT * FROM empleados;
UPDATE empleados SET departamento = 'Marketing' WHERE id = 1;
DELETE FROM empleados WHERE id = 1;
```

¿Por qué usar Tablespaces?

Los tablespaces son útiles por varias razones:

- **Organización:** Permiten organizar los datos de manera lógica y física.
- **Manejo del Espacio:** Facilitan el manejo del espacio de almacenamiento, permitiendo definir tamaños, crecimiento y límites específicos.
- **Rendimiento:** Pueden mejorar el rendimiento, ya que permiten separar físicamente los datos que son accedidos frecuentemente de los que no.
- **Seguridad y Backup:** Permiten implementar estrategias de seguridad y backup/restauraciones más efectivas, al poder enfocar estos procesos en tablespaces específicos.

3.5 Roles

En el contexto de las bases de datos, los roles son un conjunto de permisos y privilegios asignados a los usuarios para controlar el acceso y las acciones que pueden realizar sobre los datos y la estructura de la base de datos. Utilizar roles es una práctica de seguridad importante que ayuda a administrar de manera efectiva los derechos de los usuarios y proteger la integridad y la privacidad de los datos.

Funciones de los roles en una base de datos

Segregación de responsabilidades: Los roles permiten dividir las tareas entre diferentes tipos de usuarios. Por ejemplo, un rol podría ser capaz de leer datos, mientras que otro podría tener permisos para modificar o eliminar datos.

Simplificación de la administración de permisos: En lugar de asignar permisos individualmente a cada usuario, un administrador puede crear roles con un conjunto específico de permisos y luego asignar esos roles a los usuarios. Esto hace que sea más fácil y rápido gestionar los permisos.

Mejora de la seguridad: Al limitar los permisos de los usuarios según sus necesidades laborales específicas, los roles ayudan a minimizar los riesgos de seguridad, como el acceso no autorizado o la manipulación de datos críticos.

Cumplimiento normativo: En muchas industrias, existen requisitos específicos sobre quién puede o no puede acceder a cierta información. Los roles ayudan a cumplir con estas regulaciones al asegurar que solo los usuarios autorizados tengan acceso a los datos sensibles.

Ejemplos de roles comunes en una base de datos

Admin: Tiene acceso completo a todas las funciones de la base de datos, incluyendo la creación de otros usuarios, asignación de roles, definición de esquemas de base de datos, etc.

Developer: Puede tener permisos para crear y modificar tablas y procedimientos almacenados, pero no necesariamente para modificar usuarios o asignar roles.

Analyst: Generalmente tiene permisos de lectura sobre una gran cantidad de datos para realizar análisis y generar informes, pero no tiene permisos para modificar esos datos.

User: Puede tener acceso solo a leer y tal vez ingresar datos específicos relacionados con su área de trabajo.

Audit: Puede tener permisos solo para ver los registros de acceso y cambios en la base de datos para propósitos de auditoría.

Implementación de roles

La implementación específica de roles puede variar según el sistema de gestión de bases de datos (DBMS) que se utilice. Por ejemplo:

Oracle: Utiliza un sistema de roles muy robusto que permite incluso roles temporales basados en la sesión.

MySQL: Permite la creación de roles y la asignación de permisos de manera granular.

PostgreSQL: También soporta roles, y estos pueden tener otros roles, permitiendo una jerarquía de roles.

Consideraciones al usar roles

Revisión periódica: Los permisos y roles deben ser revisados regularmente para asegurar que sigan siendo relevantes y seguros conforme cambian las necesidades del negocio y el personal.

Principio de menor privilegio: Es recomendable asignar a los usuarios el nivel mínimo de acceso necesario para realizar sus tareas, reduciendo así el potencial de daños en caso de abuso de privilegios o ataque externo.

Roles Fijos de Base de Datos

Los roles fijos de base de datos son predefinidos y existen en todas las bases de datos. No se pueden cambiar los permisos asignados a estos roles. Algunos de los roles fijos más comunes incluyen:

db_owner: Permite realizar todas las actividades de configuración y mantenimiento en la base de datos, incluyendo la eliminación de la base de datos.

db_securityadmin: Permite modificar la pertenencia a roles y administrar permisos, pero solo para roles personalizados.

db_accessadmin: Permite agregar o eliminar el acceso a la base de datos para inicios de sesión de Windows, grupos de Windows e inicios de sesión de SQL Server.

db_backupoperator: Permite crear copias de seguridad de la base de datos.

db_ddladmin: Permite ejecutar cualquier comando del lenguaje de definición de datos (DDL) en una base de datos.

db_datawriter: Permite agregar, eliminar o cambiar datos en todas las tablas de usuario.

db_datareader: Permite leer todos los datos de todas las tablas y vistas de usuario.

db_denydatawriter: No permite agregar, modificar ni eliminar datos de tablas de usuario de una base de datos.

db_denydatareader: No permite leer datos de las tablas y vistas de usuario dentro de una base de datos.

Trabajando con Roles de Nivel de Base de Datos

Para trabajar con roles de nivel de base de datos, puedes utilizar varios comandos y funciones, como:

sp_helpdbfixedrole: Devuelve la lista de los roles fijos de base de datos.

sp_dbfixedrolepermission: Muestra los permisos de un rol fijo de base de datos.

sp_helprole: Devuelve información acerca de los roles de la base de datos actual.

sp_helprolemember: Devuelve información acerca de los miembros de un rol de la base de datos actual.

CREATE ROLE: Crea un nuevo rol de base de datos.

ALTER ROLE: Cambia el nombre o la pertenencia de un rol de base de datos.

DROP ROLE: Quita un rol de la base de datos.

sp_addrole: Crea un nuevo rol de base de datos.

sp_droprole: Quita un rol de la base de datos actual.

sp_addrolemember: Agrega un usuario, rol, inicio de sesión de Windows o grupo de Windows a un rol de base de datos.

sp_droprolemember: Quita una cuenta de seguridad de un rol de SQL Server de la base de datos actual.

GRANT, **DENY**, **REVOKE**: Permisos para agregar, denegar o quitar permisos a un rol.

Ejemplo en PostgreSQL

1. Crear un Rol

```
1  CREATE ROLE analista;
```

2. Asignar permisos al Rol

```
1  GRANT SELECT ON TABLE clientes TO analista;
2  GRANT SELECT ON TABLE ventas TO analista;
```

3. Crear un Usuario y asignarle el Rol

```
1  CREATE USER juan WITH PASSWORD 'password';
2  GRANT analista TO juan;
```

4. Revocar permisos y eliminar Rol

```
1  REVOKE SELECT ON TABLE clientes FROM analista;
2  REVOKE SELECT ON TABLE ventas FROM analista;
3  DROP ROLE analista;
```

Ejemplo en MySQL

MySQL maneja roles de manera similar, aunque el soporte completo de roles fue introducido a partir de MySQL 8.0.

1. Crear un Rol

```
1  CREATE ROLE 'analista';
```

2. Asignar permisos al Rol

```
1  GRANT SELECT ON dbclientes.clientes TO 'analista';
2  GRANT SELECT ON dbventas.ventas TO 'analista';
```

3. Crear un Usuario y asignarle el Rol

```
1  CREATE USER 'juan'@'localhost' IDENTIFIED BY 'password';
2  GRANT 'analista' TO 'juan'@'localhost';
```

4. Asignar el Rol al Usuario durante la sesión

```
1  SET ROLE 'analista';
```

5. Revocar permisos y eliminar Rol

```
1  REVOKE SELECT ON dbclientes.clientes FROM 'analista';
2  REVOKE SELECT ON dbventas.ventas FROM 'analista';
3  DROP ROLE 'analista';
```